Monitoring SuperClusters

Pitfalls and Practical Methodologies

Joshi Fullop
Cyber Infrastructure Group
National Center for Supercomputing Applications

The Proposition

- Provide an comprehensive set of data
- Provide data frequently
- Provide at virtually zero impact to cluster
- Be completely fault tolerant

User/Consumer Archetypes

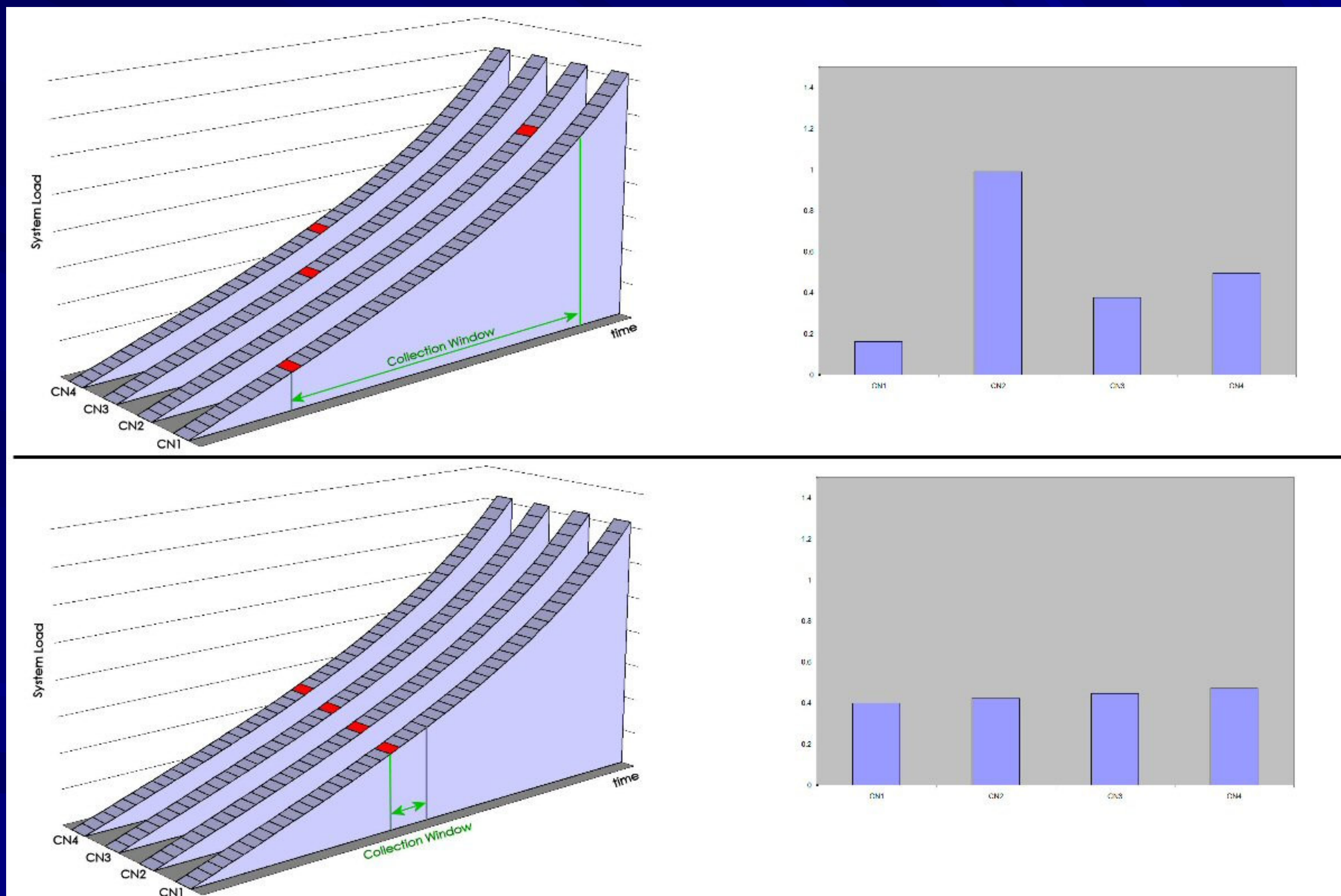
- System Administrators
- Scientists/Users
- Application Developers
- Automated tools
- Site Administration/Business Analysis

Data Collection

- Scheduler data
- Compute Node metrics
- Storage systems
- Networks

- Correlation of different types of data is key

Timeliness of Data

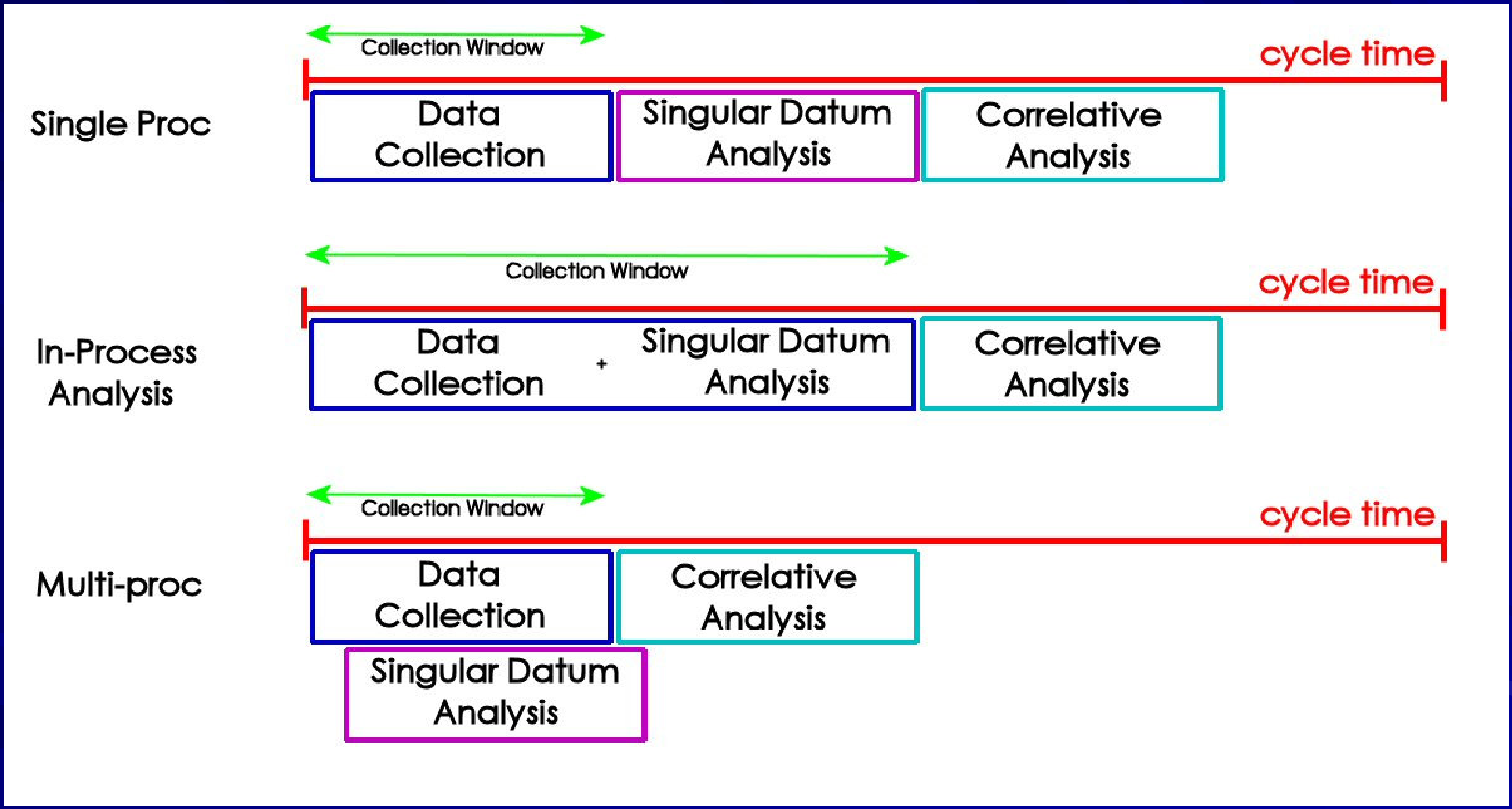


Cycle Phases

- Data Collection
  - Gathering the raw data into one place
- Singular Datum Analysis
  - Thresholds, Constants comparisons
  - Requiring that datum only and no others from the collection
- Correlative Analysis
  - Comparison of different sets of data to each other
  - Cannot be done until data collection is complete

ALL MUST SCALE

Data Collection & Analysis



Two Schools of Thought

- Synchronous
  - Central point gathers data by polling each node
  - Can be done in parallel
  - Synchronized submission becomes unstable
- Asynchronous
  - Data is submitted from each node to a central point
  - Naturally parallel, but not effectively controllable
  - Asynchronous polling is just random

Sync vs Async

- Synchronous
  - Data Collection: Controllable use of resources, easily administrated, can be done in parallel. Bound by hardware capabilities
  - Singular Datum Analysis: Can be done afterward data collection or in parallel. Can be done in-process or out of process
  - Correlative Analysis: Easily done with the data snapshot.

Sync vs Async

- Asynchronous
  - Data collection: Naturally parallel, difficult to control timing and resource usage. Requires larger software footprint on the compute node. Large potential for network storms as hardware limits are reached.
  - Singular Datum Analysis: Only easily done in-process
  - Correlative Analysis: $O(n^2)$ complexity unless some snapshot mechanism is used.

Sync vs Async

- Data Collection: Synchronous
  - Far more controllable and stable at hardware limits
- Singular Datum Analysis: Synchronous
  - Least impact on compute nodes
  - Can be done in setwise to benefit from indexes
- Correlative Analysis: Synchronous
  - Effected once per collection cycle naturally
  - Async is O(n2) without snapshot cache.
  - Asynch with snapshot cache is just like Synch, but without the benefits of the previous two sections

Pitfalls

- Unregulated Access to Core Components
  - Problem: Underlying software need protection
  - Pitfall: Pass-through information providers or 'information switchboards' allow the hammering of schedulers and such by resolving requests on each request. Depending on implementation, one request for correlated information has a multiplicative effect on the requests sent out to the resource.
  - Solution: Use cached data or data snapshots

Pitfalls

- Introducing Jitter
  - Problem: Even the smallest delays introduced to an application running on a compute node can have a cascading effect on the degradation of that application's performance
  - Pitfall: Self analysis of data on compute nodes. Often done in order to offload some of the analysis work. This can kill application performance, especially in highly communicative and long-running codes.
  - Solution: Compute node data provision services at the absolute minimum. Avoid launching new process as that is very intrusive

Pitfalls

- Network Storms
  - Problem: Unregulated network utilization can lead to flooding, mass collisions and data loss.
  - Pitfall: Some approached implement a ‘submit data on threshold change’. This sounds good for the base case, but induces floods at critical points like job launch and cleanup. This approach also increases jitter.
  - Solution: Use a synchronous collection method as asynchronous methods can naturally build to storms over time.

Pitfalls

- Fault Intolerance
  - Problem: Monitoring systems must persist when everything else fails. System must also handle unforeseen problems gracefully.
  - Pitfall: Persistent codes. Running for long periods of time increase the odds of eventually crashing.
  - Solution: Segment processes into a persistent process that just launches and monitors the collection and analysis processes.

Pitfalls

- Inflexible Data Structures
  - Problem: When software and systems that are monitored change, underlying codes and data structures must be rewritten. This leads to downtime and rushed coding.
  - Pitfall: The first thing every endeavor seems to do is create some specific schema.
  - Solution: Use a flexible column (i.e. field) oriented design.

Row-Oriented

Snapshot	Node#	%CPU	%MEM	%Idle	IRQ	....	....	....
12/25/06 20:48	1	75	55		10			

Column-Oriented

Snapshot	Node#	ParamName	ParamValue
12/25/06 20:48	1	%CPU	75
12/25/06 20:48	1	%MEM	55
12/25/06 20:48	1	IRQ	10

Pitfalls

- Detecting Discrepancies
  - Problem: When monitored systems have errors, they often misreport their status or are in a problematic state that needs to be identified.
  - Pitfall: Trusting a single source for information. Example: the scheduler indicates that a job is running on a set of nodes, but there is no activity or expected processes running on those nodes.
  - Solution: Cross-reference various data sets to look for problematic states. Systems administrators can often help identify these states.

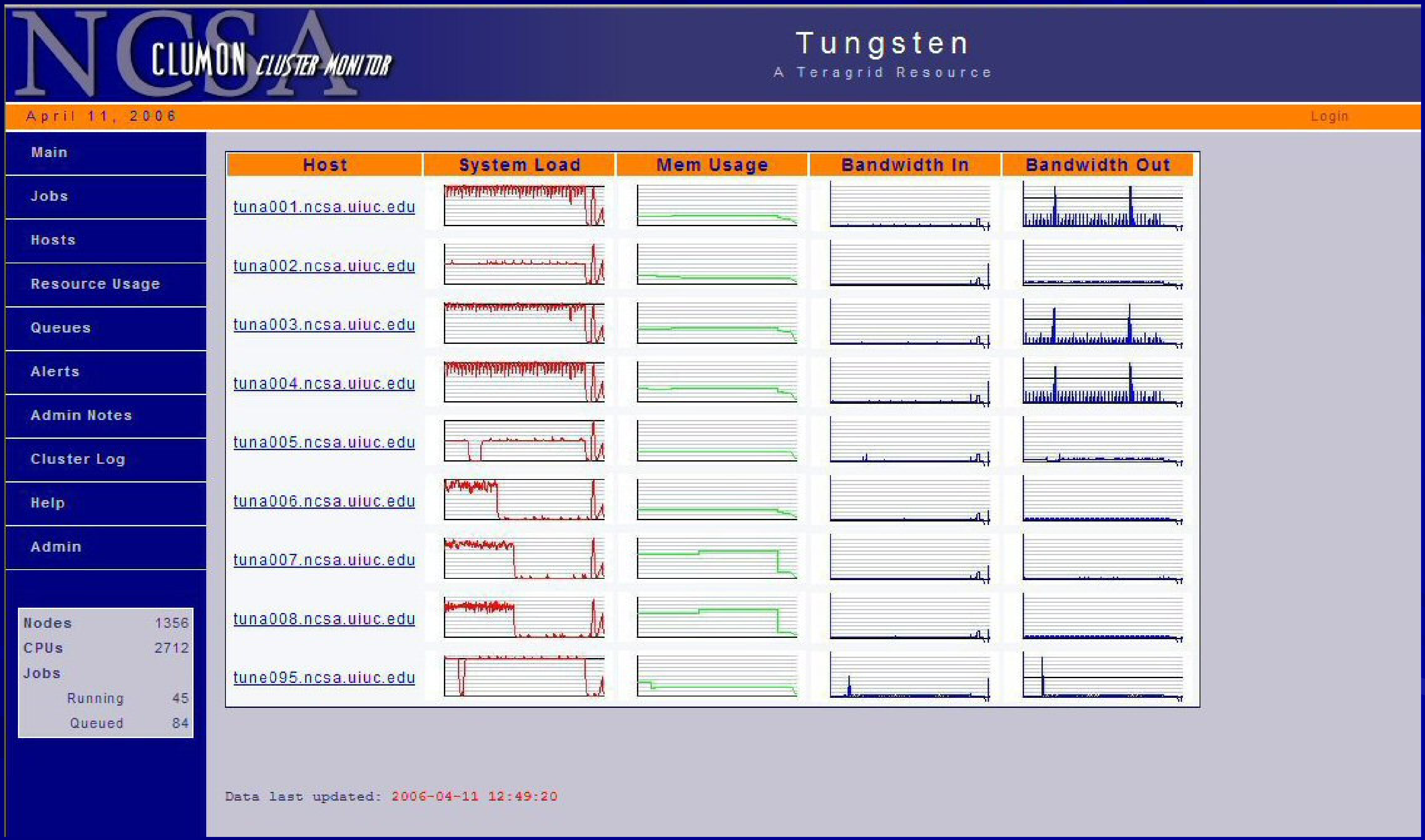
Pitfalls

- Shared or External Data Storage
  - Problem: Performance is key to being able to scale. Introduction of latency kills performance.
  - Pitfall: Co-location or using an existing communal data storage service. Monitoring does not use a database in the same manner that a run-of-the-mill database application does.
  - Solution: Utilize a dedicated data system located on the same machine as the monitoring processes. Additionally, do not co-locate other services, like the scheduler on the same machine as the monitor

Data Dissemination

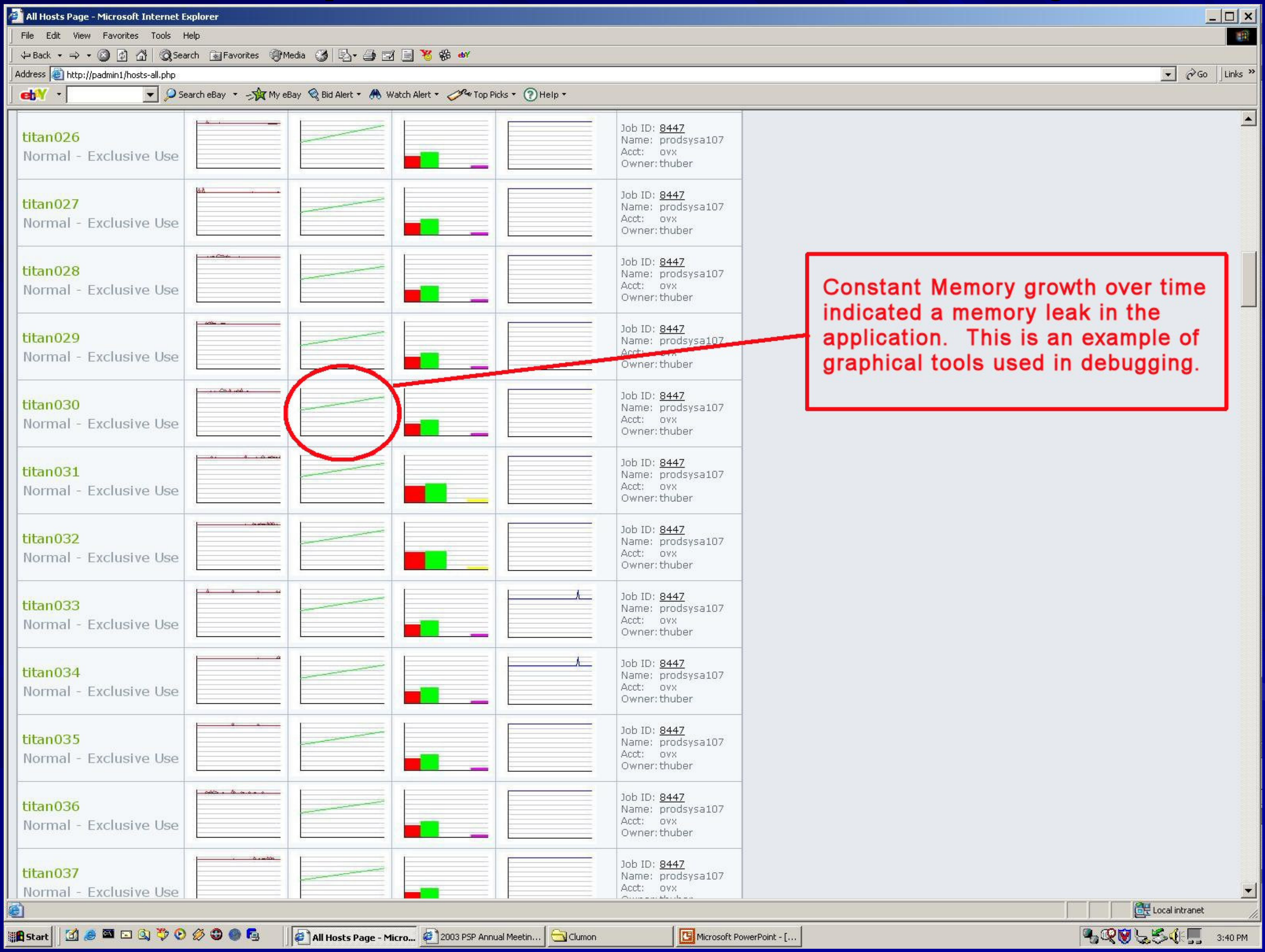
- Caching
  - Building highly correlative views of the data can adversely impact monitoring performance. Cache the common requests.
- Historical Data
  - Trend analysis is a major need and provides for greater understanding of how things work together
- Archiving
  - Monitoring deals in a great amount of data, at some point it needs summarized and off-loaded.

Job Profile

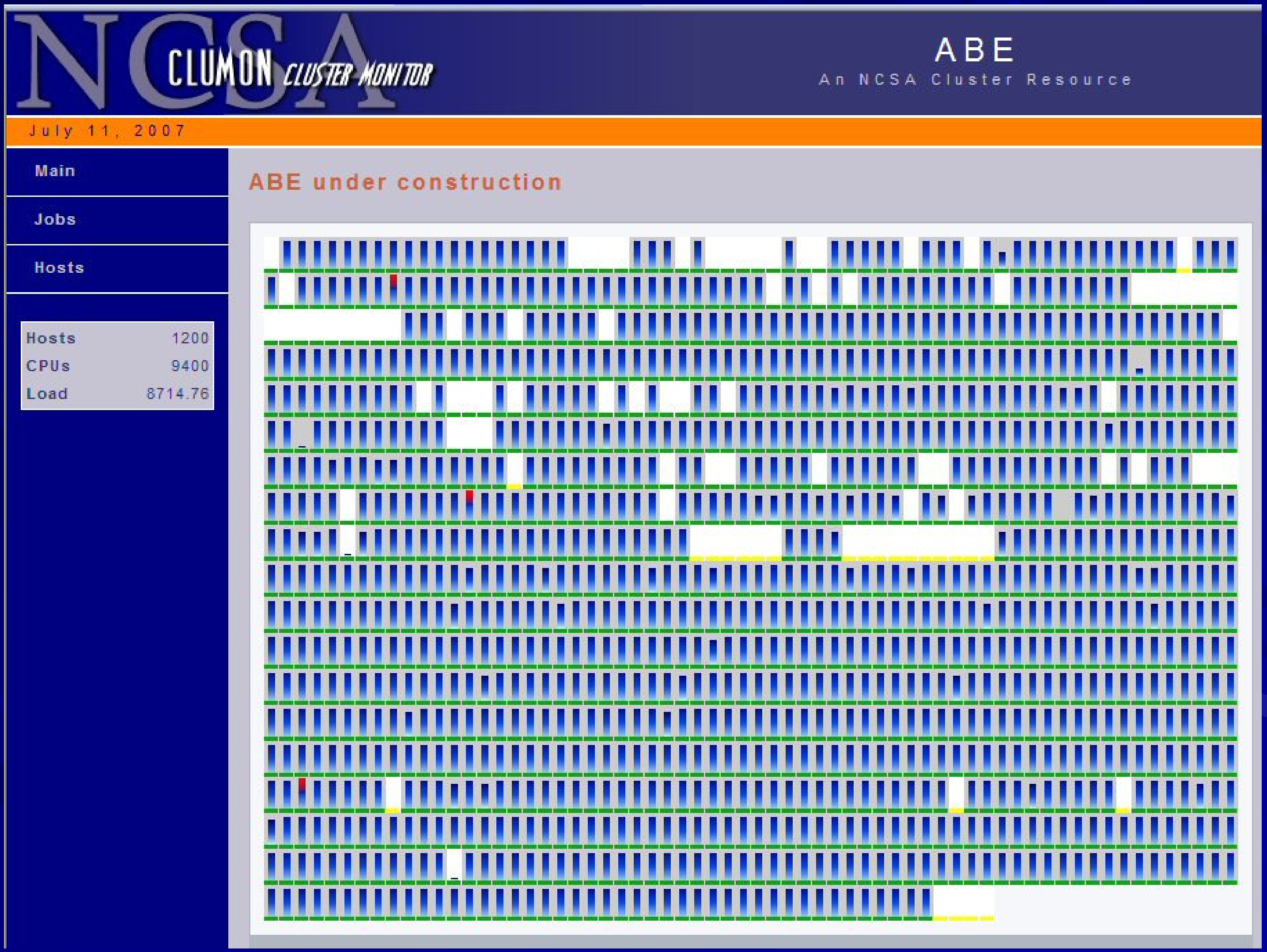


NCSA CLUMON CLUSTER MONITOR
Tungsten
A Teragrid Resource
April 11, 2006
Login
Main
Jobs
Hosts
Resource Usage
Queues
Alerts
Admin Notes
Cluster Log
Help
Admin
Nodes 1356
CPUs 2712
Jobs
Running 45
Queued 84
Host
System Load
Mem Usage
Bandwidth In
Bandwidth Out
tuna001.ncsa.uiuc.edu
tuna002.ncsa.uiuc.edu
tuna003.ncsa.uiuc.edu
tuna004.ncsa.uiuc.edu
tuna005.ncsa.uiuc.edu
tuna006.ncsa.uiuc.edu
tuna007.ncsa.uiuc.edu
tuna008.ncsa.uiuc.edu
tune095.ncsa.uiuc.edu
Data last updated: 2006-04-11 12:49:20

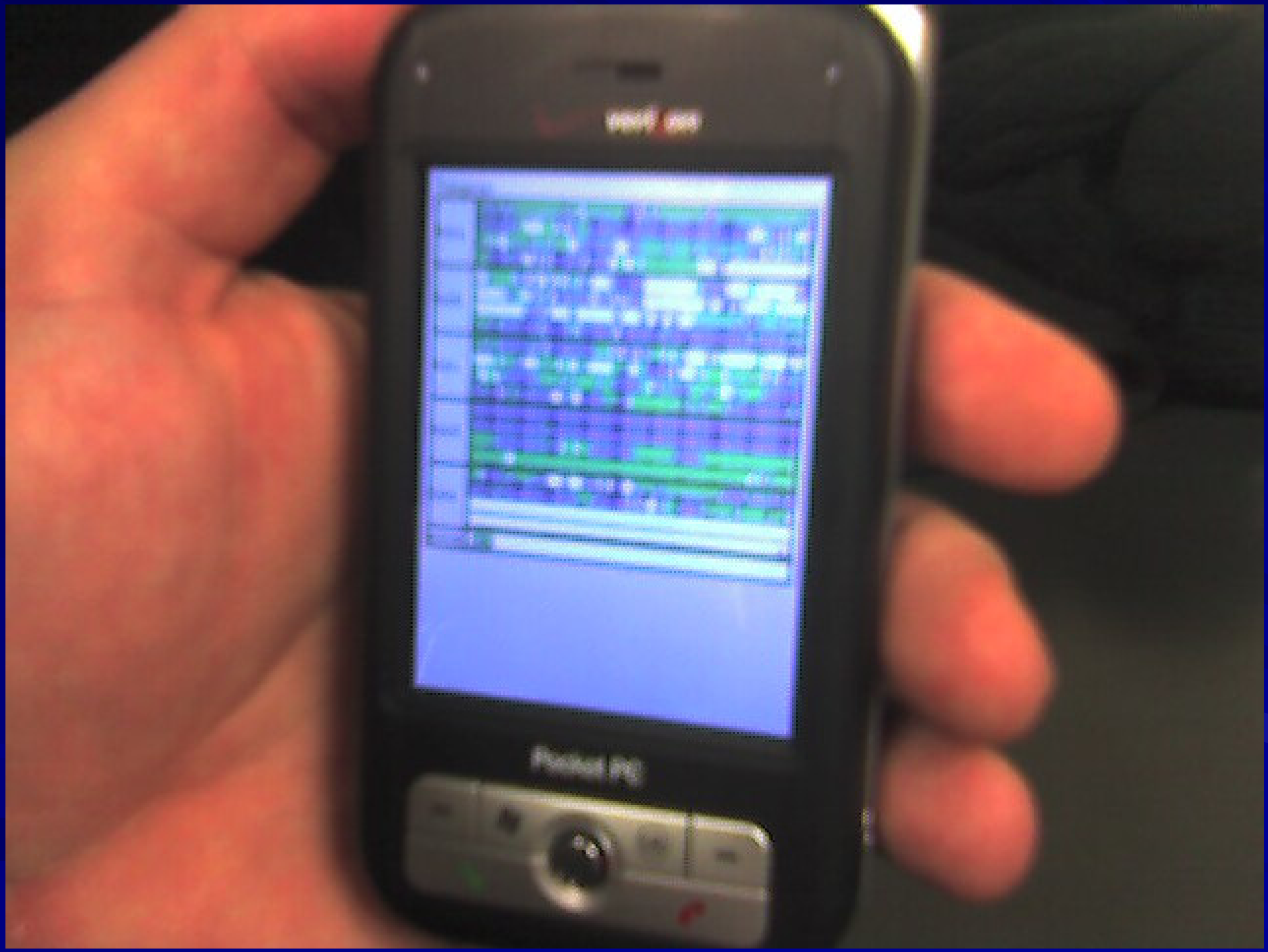
Example of Trend Analysis



Fragmentation



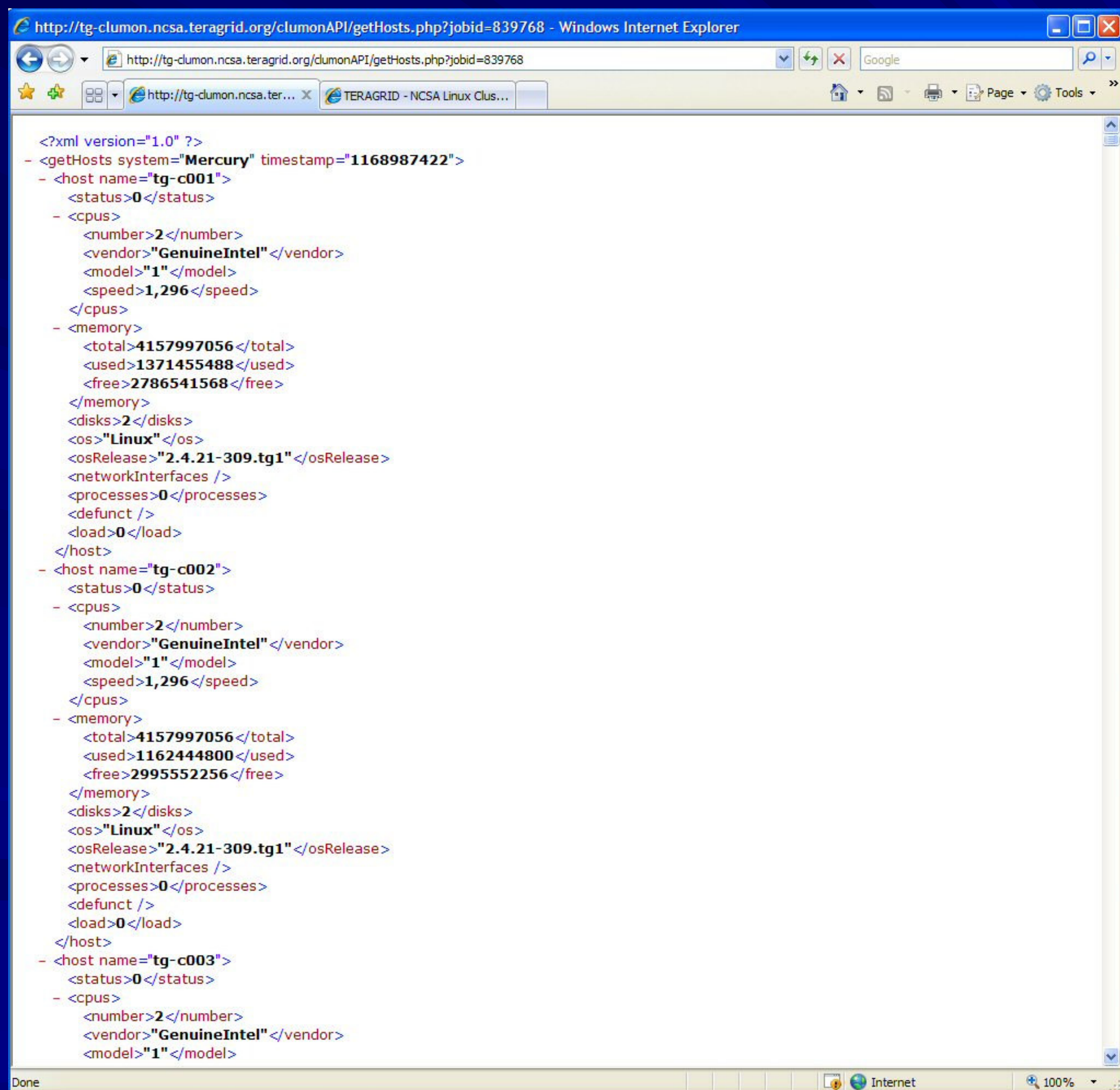
Data Density



Automation

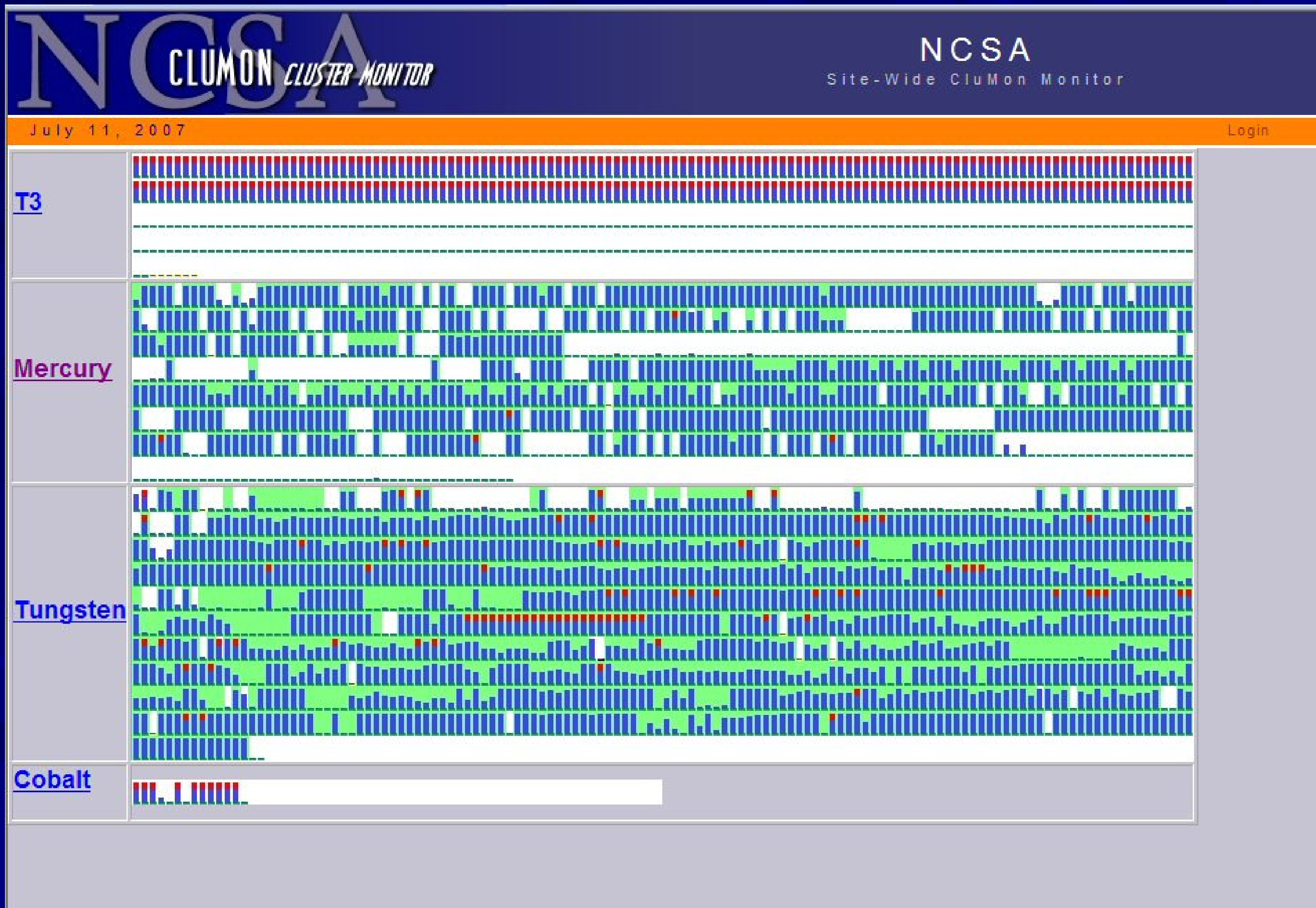
- Componentized Design
- Web-based API
- Event Notification System

Web-based API




```xml
<?xml version="1.0" ?>
<getHosts system="Mercury" timestamp="1168987422">
  <host name="tg-c001">
    <status>0</status>
    <cpus>
      <number>2</number>
      <vendor>"GenuineIntel"</vendor>
      <model>"1"</model>
      <speed>1,296</speed>
    </cpus>
    <memory>
      <total>4157997056</total>
      <used>1371455488</used>
      <free>2786541568</free>
    </memory>
    <disks>2</disks>
    <os>"Linux"</os>
    <osRelease>"2.4.21-309.tg1"</osRelease>
    <networkInterfaces />
    <processes>0</processes>
    <defunct />
    <load>0</load>
  </host>
  <host name="tg-c002">
    <status>0</status>
    <cpus>
      <number>2</number>
      <vendor>"GenuineIntel"</vendor>
      <model>"1"</model>
      <speed>1,296</speed>
    </cpus>
    <memory>
      <total>4157997056</total>
      <used>1162444800</used>
      <free>2995552256</free>
    </memory>
    <disks>2</disks>
    <os>"Linux"</os>
    <osRelease>"2.4.21-309.tg1"</osRelease>
    <networkInterfaces />
    <processes>0</processes>
    <defunct />
    <load>0</load>
  </host>
  <host name="tg-c003">
    <status>0</status>
    <cpus>
      <number>2</number>
      <vendor>"GenuineIntel"</vendor>
      <model>"1"</model>
```

Multi-Site Monitoring



Future Systems